VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZŠÍŘENÍ PODPORY ARCHIVŮ V SYSTÉMU GVFS

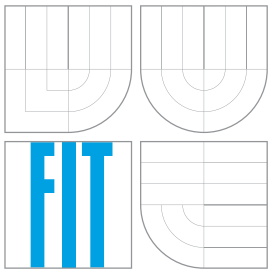
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ HOLÝ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZŠÍŘENÍ PODPORY ARCHIVŮ V SYSTÉMU GVFS

AN EXTENSION OF SUPPORT FOR ARCHIVES IN GVFS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ HOLÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2012

Abstrakt

Tato práce se zabývá rozšířením démona pro manipulaci s archivy v systému GVFS. Na základě analýzy současného stavu démona, možností knihovny LibArchive a podobných projektů bylo navrženo, implementováno a otestováno rozšíření. Hlavním přínosem tohoto rozšíření je snadná práce se soubory uvnitř archivu pomocí GIO API, které je využíváno množstvím aplikací běžících pod operačním systémem GNU/Linux. Obsažené soubory a složky v archivu lze nyní číst, přejmenovávat, přesouvat, kopírovat nebo mazat, dále lze vytvářet prázdné adresáře. Démon také umožňuje vytvářet nové archivy požadovaného formátu.

Abstract

The bachelor thesis deals with an extension of archive daemon in GVFS system. The extension was designed, implemented, and tested based on an analysis of current state of the daemon, LibArchive library options, and similar projects. The main benefit of the extension is an easy manipulation with files in an archive via GIO API, which is used by many applications running in GNU/Linux operating system. Files and folders included in an archive can be read, renamed, moved, copied or deleted, it can also create empty directories. Daemon also allows creating new archives of a desired format.

Klíčová slova

GLib, GIO, GFile, GVFS, Archiv, LibArchive

Keywords

GLib, GIO, GFile, GVFS, Archive, LibArchive

Citace

Ondřej Holý: Rozšíření podpory archivů v systému GVFS, bakalářská práce, Brno, FIT VUT v Brně, 2012

Rozšíření podpory archivů v systému GVFS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Aleše Smrčky. Odborné konzultace probíhaly ve spolupráci s panem Tomášem Bžatkem ze společnosti Red Hat Czech, s. r. o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Holý
12. května 2012

Poděkování

Děkuji vedoucímu práce doktoru Aleši Smrčkovi za pedagogické vedení. Stejně tak děkuji konzultantovi Tomáši Bžatkovi za odbornou pomoc. Oběma jmenovaným děkuji za cenné rady a připomínky, které mi ochotně poskytovali během tvorby bakalářské práce. V neposlední řadě děkuji celé své rodině za jejich vydatnou podporu.

Obsah

Kapitola 1

Úvod

I v dnešní době, kdy se již nemusí řešit problémy s páskovými mechanikami či paměťové nároky, jsou archivy stále hojně využívány. Téměř každá domácnost má přístup k internetu, který je postaven na principu sdílení informací. Potřeba sdílet data s rozmachem sociálních sítí v poslední době ještě více vzrostla. I přes různá dodatečná rozšíření prohlížečů jsou možnosti přenosu více souborů omezené a roli tak přebírají archivy různých formátů. Uživatelům je proto potřeba zajistit co nejjednodušší manipulaci s nimi.

Jedním z nejpohodlnějších způsobů práce s archivy je jejich integrace a možnosti úprav pomocí systémového souborového manažeru. Odpadají tak problémy s různými uživatelskými rozhraními či vzájemná nekompatibilita programů při kopírování. Pro GNU/Linux existuje několik takových řešení s různou funkcionalitou. Jedním z nich je právě systém *GVFS* (viz sekce 2.2), který umožňuje připojení různých virtuálních svazků a jednotnou práci skrze *GIO API* (viz sekce 2.1.2). Toto aplikační rozhraní využívá spousta aplikací grafického prostředí plochy *GNOME*[1] včetně několika správců souborů.

Démon pro podporu archivů v systému GVFS je postaven na knihovně *LibArchive* (viz sekce 2.3) a bohužel v současné době umožňuje pouze čtení obsahu archivů. Tato skutečnost již pár let trápí některé jeho uživatele [1, 2], a proto vznikl požadavek na implementaci nové funkcionality [3]. Cílem této práce je napravit tento nedostatek a přispět tak komunitě a uživatelům GNU/Linuxu.

V kapitole 2 je popsán systém GIO s podsystémem GVFS, knihovna LibArchive, další používané technologie a jiná již existující řešení. Následující kapitola 3 popisuje návrh na vytvoření příslušného rozšíření pro systém GVFS. Kapitoly 4 a 5 rozebírají konkrétní implementační detaily a způsoby testování. V poslední kapitole 6 lze nalézt zhodnocení vytvořené práce a náměty na další vývoj.

[1] http://www.gnome.org/

Kapitola 2

Použité principy a technologie

Tato kapitola přibližuje systém GIO, podsystém GVFS, démona pro práci s archivy, knihovnu LibArchive a další použité technologie. Popisuje princip jejich činnosti a způsob práce s aplikačním rozhraním. Nakonec jsou zmíněna jiná již existující řešení, ze kterých jsou vyvozeny určité závěry ovlivňující návrh rozšíření.

2.1 Programátorská knihovna GLib

GLib[1] je multiplatformní programátorská knihovna psaná v jazyce C [4, 5]. Do roku 2002 byla součástí projektu GTK+, před vydáním druhé verze bylo však vývojáři rozhodnuto o oddělení kódu, který nesouvisí s grafickým uživatelským rozhraním. Tento postup zajistil širší rozšíření. Knihovna funguje na většině systémů založených na Unixu, dále v Microsoft Windows, OS/2 či BeOS. Je vydávána pod svobodnou licencí GNU LGPL. Všechny knihovní funkce, kromě operací pro manipulaci s datovými strukturami, jsou navrhovány s ohledem pro paralelní běh.

GLib je dnes univerzální knihovnou poskytující užitečné datové typy, makra, typové konverze, funkce pro práci s řetězci, operace se soubory, práci s hlavní programovou smyčkou, práce s pamětí aj. Nabízí pokročilé datové struktury, jako jsou např. seznamy, hašovací tabulky, binární stromy, dynamická pole či řetězce.

Knihovna GLib se skládá z několika samostatných částí:

- *GObject* – univerzální objektový systém pro jazyk C
- *GIO* – práce se soubory a V/V operacemi
- *GModule* – dynamické načítání objektových souborů
- *GThreads* – abstrakce pro vlákna a potřebné nástroje pro synchronizaci
- *GLib* – ostatní funkce (např. pro práci s řetězci, pamětí či chybami)

2.1.1 Objektový systém GObject

GLib Object System, krátce *GObject*, je objektově orientovaný systém pro jazyk C [6, 7]. Jeho funkce umožňují, i přes použitý programovací jazyk, využívat výhod objektově orientovaných jazyků. Přináší třídní typový systém s jednoduchou dědičností, funkce pro správu

[1] http://developer.gnome.org/glib/

paměti, implementace základních datových typů, komunikaci pomocí signálů atd. GObject nepřidává žádné nové syntaktické konstrukce, využívá především makra. Díky tomu je závislý pouze na ostatních částech knihovny GLib a standardní knihovně jazyka C. Je na něm postavena většina aplikací a knihoven grafického prostředí Gnome.

Typový systém GType Základ objektového systému GObject tvoří typový systém *GType*, který obsahuje běhové informace o typech [8]. Má prostředky pro vyváření, kopírování, přiřazování a odstraňování jednoduchých typů i složitých objektů. GType lze jednoduše provázat s dalšími programovacími jazyky, díky tomu je celý systém GObject jednoduše mapovatelný do objektově orientovaných systémů různých programovacích jazyků (např. C++, .NET, Ruby či Python).

2.1.2 Abstrakce souborového systému GIO

Další částí knihovny GLib je systém *GIO*, který poskytuje API pro práci se souborovým systémem [9, 10, 11]. Na rozdíl od svého předchůdce, systému *GnomeVFS*[2], poskytuje moderní a jednoduché aplikační rozhraní. Nejedná se totiž o pouhou kopii POSIX API, ale s využitím objektového návrhu poskytuje vývojářům mnohem vyšší úroveň abstrakce.

V základu GIO pracuje pouze s lokálním souborovým systémem. Ve spojení s podsystémem *GVFS*, které je rozebráno v sekci 2.2, lze pomocí stejného API přistupovat např. k různým síťovým souborovým systémům. Toho využívají zejména správci souborů jako *Nautilus*[3], *Thunar*[4] či *PCManFM*[5].

GIO obsahuje množství funkcí a tříd pro práci se soubory, pro tuto práci jsou důležité zejména následující:

- `GFile` – abstraktní reprezentace souboru
- `GFileInfo` – informace o souboru či souborovém systému
- `GFileEnumerator` – seznam souborů v adresáři
- `GMount` – reprezentace připojitelného souborového systému

Přenos dat je realizován pomocí datových toků, najdeme zde tedy i několik tříd pro jejich zapouzdření:

- `GFileInputStream` – čtení dat ze souboru
- `GFileOutputStream` – zápis dat do souboru
- `GFileIOStream` – čtení i zápis dat souboru

V GIO je samozřejmě spousta dalších tříd např. pro uchovávání nastavení aplikací, pro síťové programování, komunikaci přes D-Bus. Kromě práce se soubory a V/V operacemi GIO dále poskytuje možnosti pro monitorování souborů, asynchronní operace či doplňování názvů souborů.

[2] http://developer.gnome.org/gnome-vfs/
[3] http://live.gnome.org/Nautilus
[4] http://thunar.xfce.org/
[5] http://pcmanfm.sourceforge.net/

2.1.3 Abstrakce souboru GFile

Třída `GFile` je abstrakcí souboru [12]. Instance této třídy však nereprezentují soubory samotné, ale jedná se pouze o jednoznačný identifikátor souboru. Všechny V/V operace jsou realizovány pomocí příslušných tříd. Nový objekt lze vytvořit pomocí cesty, ať už absolutní či relativní, nebo pomocí URI. Umístění pro soubory na disku začínají schématem `file://`.

Pro práci s objektem `GFile` můžeme využít synchronní i asynchronní metody. Následuje výčet nejdůležitějších synchronních funkcí:

- `g_file_query_filesystem_info` – získání informací o souborovém systému
- `g_file_make_directory` – vytvoření nového adresáře
- `g_file_enumerate_children` – získání seznamu souborů v adresáři
- `g_file_query_info` – získání informací o souboru či adresáři
- `g_file_set_display_name` – přejmenování souboru či adresáře
- `g_file_delete` – odstranění souboru či adresáře
- `g_file_copy` – kopírování souboru či adresáře
- `g_file_move` – přesunutí souboru či adresáře
- `g_file_read` – čtení obsahu souboru
- `g_file_append_to`, `g_file_create`, `g_file_replace` – zápis dat do souboru

Virtuální souborové systémy poskytované systémem GVFS musí být nejprve připojeny. Pro připojení a odpojení jsou zde asynchronní funkce `g_file_mount_enclosing_volume` a `g_file_unmount_mountable`, které pracují s objektem `GMount`.

Metody pro práci s objektem `GFile` mají ustálené pořadí parametrů. Prvním je soubor, se kterým se pracuje, následují dodatečné parametry k provedení operace a na závěr objekty `GCancellable` a `GError`. Zmíněné objekty budou blíže popsány dále.

Přerušení operace pomocí GCancellable Třída `GCancellable` poskytuje mechanismus pro přerušení prováděné operace [13]. Je navržena s ohledem na vícevláknové aplikace a poskytuje jednotný způsob přerušení pro asynchronní i synchronní operace GIO. Typickým použitím je napojení na tlačítko pro zrušení prováděné akce v souborových manažerech (např. při nechtěném spuštění nebo při dlouhém provádění).

Sledování postupu pomocí GFileProgressCallback Některé metody mají jako jeden z parametrů objekt `GFileProgressCallback`. S jeho pomocí lze poskytovat informace o probíhající operaci, kolik dat již bylo zpracováno a kolik jich je celkem.

2.1.4 Zpracování chyb pomocí GError

Jako většina neobjektově orientovaných jazyků ani jazyk C nenabízí systém výjimek. Standardní metody pro oznamování chyb z volané funkce poskytuje knihovna GLib pomocí objektu `GError` [14]. Ten se používá pro obnovitelné běhové chyby a je obvykle uváděn jako poslední parametr volané funkce.

Třída `GError` obsahuje tři části:

- *doménu* – umístění funkce oznamující chybu
- *kód* – číselné označení konkrétní chyby
- *zprávu* – textová zpráva popisující chybu

Pro oznamování chyb systému GIO se používá doména `G_IO_ERROR` a příslušné chybové kódy typu `enum GIOErrorEnum` [15]. Podstatné pro tuto práci jsou zejména:

- `G_IO_ERROR_FAILED` – obecná chyba V/V operace
- `G_IO_ERROR_BUSY` – právě probíhá jiná V/V operace
- `G_IO_ERROR_INVALID_FILENAME` a `G_IO_ERROR_INVALID_ARGUMENT` – špatné argumenty
- `G_IO_ERROR_NOT_FOUND` a `G_IO_ERROR_EXISTS` – požadovaný soubor či adresář (ne)existuje
- `G_IO_ERROR_WOULD_MERGE` – došlo by ke slučování adresářů
- `G_IO_ERROR_WOULD_RECURSE` – operace s adresáři by byla rekurzivní
- `G_IO_ERROR_IS_DIRECTORY` a `G_IO_ERROR_NOT_DIRECTORY` – (ne)jedná se o adresář

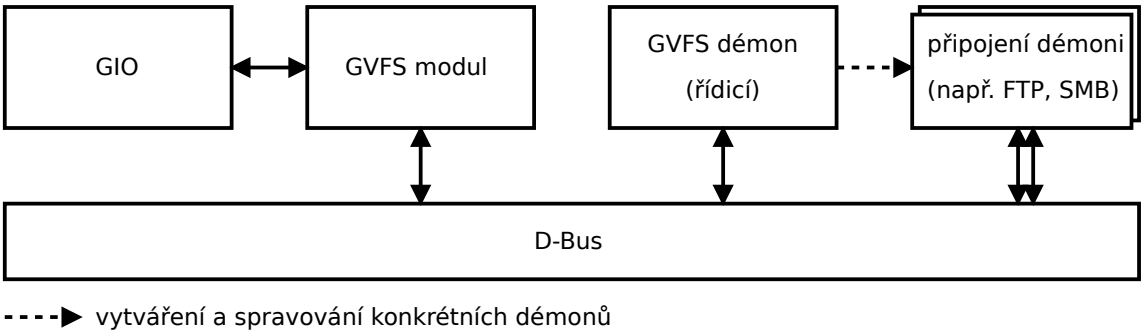
Překlady pomocí nástroje GetText Popis chybové zprávy je uváděn v angličtině. Aby se však k uživateli dostal vhodný překlad, zapisuje se pomocí makra „_“ [16]. Za běhu jsou pak tyto zprávy nahrazeny nástrojem `gettext`[6] podle uživatelem nastaveného jazyka v systému.

2.2 Virtuální souborový systém GVFS

GVFS rozšiřuje možnosti GIO o virtuální souborové systémy. Mezi nimi jsou různé síťové protokoly (mj. FTP, SFTP, ObexFTP, SMB, WebDAV), speciální systémová místa (např. koš, připojené svazky), práce s archivy apod. [17, 18, 19, 20]. K těmto umístěním lze přistupovat přes speciální URI. Například pro přístup na FTP server zabezpečený přihlašovacími údaji bude identifikátor ve tvaru `ftp://login:heslo@ftp.domena/cesta`. Pro aplikace, které neumí pracovat s těmito adresami, může být volitelně využíván nástroj *FUSE*[7]. Poté lze přistupovat k požadovaným souborům ve složce `~/.gvfs/` pomocí klasických POSIX funkcí.

Princip vzájemné komunikace je naznačen na obrázku 2.1. Pro speciální URI předá GIO požadavek GVFS modulu, který informaci pošle na datovou sběrnici *D-Bus*[8] hlavnímu démonu. Tento hlavní démon spouští a spravuje jednotlivé potřebné démony. Ty běží samostatně mimo hlavní proces, což minimalizuje nebezpečí pádu celé aplikace a dělá tak systém robustnější. Prováděná operace je v systému GVFS reprezentována objektem `GVfsJob`. Tento objekt obsahuje informace o typu operace, o jejím dokončení, o chybách, o přerušení atd.

[6] http://www.gnu.org/software/gettext/
[7] http://fuse.sourceforge.net/
[8] http://dbus.freedesktop.org/



Obrázek 2.1: Komunikace systému GIO s GVFS démony.

Díky komunikaci realizované pomocí datových toků není problém pracovat naráz s více démony. Například je možné si na vzdáleném serveru prohlížet obsah archivu a z něho kopírovat soubory třeba na paměťovou kartu v mobilu. Další výhodou je, že V/V model GIO je stavový. Pro uživatele to mimo jiné znamená, že heslo k připojení na SFTP server stačí zadat jen jednou a všechny aplikace v daném sezení mohou přistupovat k tomuto souborovému systému bez opětovného zadávání hesla.

Aplikační rozhraní démona Démon je objektem, který dědí ze třídy `GVfsBackend`. Rodičovská třída obsahuje deklarace metod pro práci s daty. Všechny funkce jsou poskytovány ve dvou verzích, lišících se prefixem. Předpona `try_` označuje neblokující rychlé operace, které čtou např. pouze z dočasné paměti a lze je vykonat ihned. Metody bez prefixu by měly být spouštěny v samostatném vlákně, protože jejich provádění může být delší. Systém GVFS se vždy primárně snaží využívat rychlejší verze, pokud jsou implementované.

Konfigurační soubor démona Pokud GIO nedokáže samo URI zpracovat, předá ho systému GVFS. Každý démon má zaregistrovaný vlastní identifikátor, který má uložen v konfiguračním souboru. Příklad takového konfiguračního souboru `archive.mount` lze vidět na obrázku 2.2. Podstatná je především položka `Scheme` popisující použité URI.

```
[Mount]
Type=archive
Exec=@libexecdir@/gvfsd-archive
AutoMount=false
Scheme=archive
```

Obrázek 2.2: Konfigurační soubor démona pro práci s archivy.

2.2.1 Nástroje pro práci s GVFS

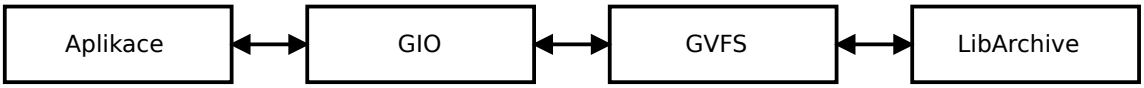
Kromě aplikačního rozhraní knihovny GIO obsahuje systém GVFS pomocné příkazy pro manipulaci s virtuálními souborovými systémy. Užitečné jsou zejména [21]:

- `gvfs-mount` – připojování či odpojování souborových systémů
- `gvfs-mkdir` – vytvoření nového adresáře

- `gvfs-ls` – výpis obsahu adresáře
- `gvfs-info` – informace o souboru či adresáři
- `gvfs-copy` – kopírování souboru či adresáře
- `gvfs-move` – přesunutí souboru či adresáře
- `gvfs-rm` – odstranění souboru či adresáře
- `gvfs-cat` – tisk obsahu souboru na standardní výstup

2.2.2 Démon pro práci s archivy

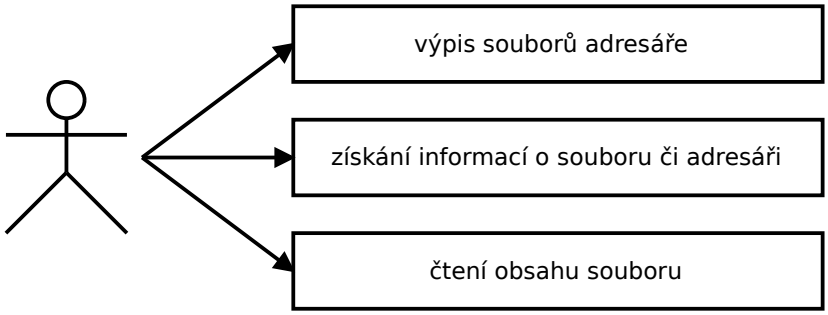
Součástí systému GVFS je od roku 2008 démon pro práci s archivy. Ke své činnosti využívá funkce knihovny *LibArchive*, která je blíže popsána v sekci 2.3. Komunikace mezi jednotlivými součástmi je znázorněna na obrázku 2.3. Démon pro práci s archivy slouží jako prostředník mezi knihovnou LibArchive a systémem GIO, jehož programátorské rozhraní využívají různé aplikace. Démon má zaregistrované schéma URI `archive://`. Veškeré zdrojové kódy démona jsou v souborech `gvfsbackendarchive.h` a `gvfsbackendarchive.c`.



Obrázek 2.3: Komunikace jednotlivých součástí.

Princip činnosti démona je následující. Po připojení archivu přečte jeho hlavičky a vytvoří si virtuální strom s informacemi o souborech. Proto jsou operace pro výpis adresáře či získání informací o souborech velmi rychlé. Naproti tomu se při požadavku na čtení musí znovu otevřít archiv, nalézt soubor a vytvořit vstupní datový tok. Po přečtení souboru je archiv opět uzavřen. Při odpojení archivu je virtuální strom a všechna ostatní data uvolněna z paměti a při příštím připojení se musí vytvářet znovu.

Démon podporuje v současné době minimum operací, což je znázorněno na obrázku 2.4. Kromě připojení a odpojení jsou implementovány funkce pro získání informací o souboru či souborovém systému, výpis souborů adresáře a čtení obsahu souboru.



Obrázek 2.4: Podporované operace démona pro práci s archivy.

Démona pro práci s archivy tvoří tři stěžejní části a s nimi spojené struktury. Jednak je to samotná třída `GVfsBackendArchive` a zbylou částí jsou pomocné funkce pro práci s archivem a virtuálním stromem. Tyto části budou blíže popsány dále.

Připojení archivu do systému GVFS

Připojit archiv do systému GVFS můžeme pomocí příkazu `gvfs-mount`, knihovních funkcí GIO nebo využitím spustitelného souboru démona `gvfsd-archive`. Adresa může vypadat jako na obrázku 2.5. V prvním případě je použitá klasická cesta k souboru. V dalších případech začátek URI tvoří již zmíněné schéma. V druhém případě následuje zadaná absolutní cesta, ve které jsou převedeny nebezpečné znaky podle *RFC 3986*[9] [22]. Ve třetím případě je použitý identifikátor souboru na lokálním disku, kde již díky způsobu zpracování musí dojít k dvojímu překódování. K samotnému převedení lze opět využít knihovní funkce (např. `g_uri_escape_string`).

```
./gvfsd-archive file="/archiv.tar"
./gvfsd-archive host="archive://%2Farchiv.tar/"
./gvfsd-archive host="archive://file%253A%252F%252F%252Farchiv.tar/"
```

Obrázek 2.5: Příklad připojení archivu do systému GVFS.

Zejména správci souborů pro jednoduché připojení archivu používají nástroj *Archive Mounter*, který zavolá spustitelný soubor démona s příslušným parametrem. Při správně nastavených asociacích souborů se po kliknutí na archiv objeví připojený nový svazek. Tento nástroj je obsažen ve většině balíčkovacích systémů distribucí GNU/Linuxu.

Třída démona GVfsBackendArchive

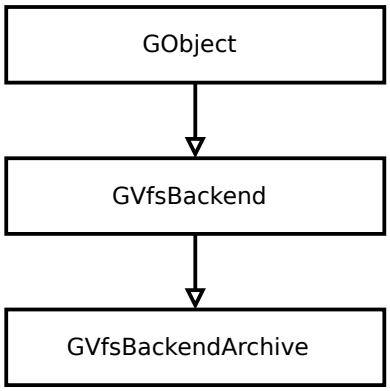
Na obrázku 2.6 je třída `GVfsBackendArchive`. Kromě instance rodičovské třídy obsahuje pouze dvě proměnné: reprezentaci souboru archivu `file` a kořen stromu souborů `files`. Z rodičovské třídy `GVfsBackend` (viz obrázek 2.7) jsou implementované následující metody (ve zdrojovém kódu démona mají předponu `do_`):

- `mount` a `unmount` – připojení a odpojení démona
- `open_for_read`, `read` a `close_read` – čtení obsahu souboru
- `enumerate` – vrácení seznamu souborů v adresáři
- `query_info` a `try_query_fs_info` – získání informací o souboru, adresáři či systému

```
typedef struct
{
  GVfsBackend  backend;
  GFile *      file;
  ArchiveFile *files;
} GVfsBackendArchive;
```

Obrázek 2.6: Třídní proměnné démona.

[9] http://www.ietf.org/rfc/rfc3986.txt



Obrázek 2.7: Hierarchie dědičnosti tříd démona pro práci s archivy.

Virtuální strom souborů

Virtuální strom souborů je pro urychlení operací nezbytný. Pro práci s ním démon obsahuje funkce, které umožňují přidání nového souboru, vyhledávání souboru ve stromu, nastavení informací o souboru nebo smazání celého stromu. Většina těchto funkcí začíná prefixem `archive_file_`. Tyto funkce využívají struktury `ArchiveFile`, která tvoří uzel virtuálního stromu (viz obrázek 2.8). Struktura obsahuje tři proměnné: jméno souboru `name`, informace o souboru `info` a seznam jeho potomků `children`.

```
typedef struct
{
  char *     name;
  GFileInfo *info;
  GSList *   children;
} ArchiveFile;
```

Obrázek 2.8: Struktura souboru pro tvorbu virtuálního stromu.

Práce s archivem

Funkce pro práci s archivem využívají struktury `GVfsArchive` znázorněné na obrázku 2.9. `GVfsArchive` obsahuje proměnné: strukturu archivu `archive`, reprezentaci souboru archivu `file`, vstupní datový tok pro realizaci přenosů `stream`, informace o operaci `job`, chybové hlášení `error` a vyrovnávací paměť pro přenosy `data`. Implementovány jsou funkce pro otevření a zavření archivu, dopředné skoky v archivu a čtení dat. Jejich názvy začínají pomocí předpony `gvfs_archive_`.

2.3 Programátorská knihovna LibArchive

LibArchive[10] je programátorská knihovna pro práci s archivy z roku 2004 [23, 24]. Původně byla součástí balíčkovacího systému LibPKG ve FreeBSD, kde bylo cílem zefektivnit práci s balíčky. Později byly s její pomocí implementovány systémové nástroje `bsdtar` a `bsdcpio`.

[10] http://libarchive.github.com/

```
typedef struct
{
  struct archive *  archive;
  GFile *           file;
  GFileInputStream *stream;
  GVfsJob *         job;
  GError *          error;
  guchar            data[4096];
} GVfsArchive;
```

Obrázek 2.9: Struktura pro práci s archivem.

Dnes se knihovna LibArchive chlubí spoustou vlastností a jedná se tak o poměrně unikátní projekt.

Stejně jako knihovna GLib je i knihovna LibArchive napsána v jazyce C. Podporuje řadu běžných formátů archivů (např. ZIP, RAR, TAR, 7-ZIP) a kompresí (mj. GZIP, BZIP2) včetně jejich automatické detekce [25]. Programátorovi tak umožňuje skrze jednotné API pracovat s různými typy souborů i přes jejich rozdílnou vnitřní strukturu. Díky dobrému návrhu je možnost přidání dalších formátů velmi jednoduchá. Knihovna je multiplatformní a lze ji tak provozovat na všech majoritních operačních systémech.

Princip činnosti vychází z formátu TAR a použití na páskových mechanikách. Abstraktní schéma struktury archivu je znázorněno na obrázku 2.10. V základu knihovna pracuje sekvenčně a v datovém toku se nevrací zpět. Pro optimalizace je však možné zapnout funkce pro skákání v datovém toku, pokud to situace povoluje. Aplikační rozhraní je striktně rozděleno na čtení a zápis. Při čtení se sekvenčně prochází hlavičky a data souborů, při zápisu se naopak postupně tyto hlavičky a data zapisují. Rozdělené API pro čtení či zápis a navržená lineární abstrakce archivu je důvodem, proč knihovna nepodporuje modifikace a náhodný přístup [26].

Obrázek 2.10: Schéma struktury archivu TAR.

Knihovna ZLib LibArchive obsahuje kompletní implementace algoritmů pro většinu formátů, nicméně pro formáty ZIP a GZIP využívá funkce knihovny *ZLib*[11]. ZLib je svobodná, otevřená a multiplatformní programátorská knihovna napsaná v jazyce C, která poskytuje abstrakci pro kompresní algoritmus Deflate.

2.3.1 Aplikační rozhraní LibArchive

I přes použitý programovací jazyk by se dalo říct, že má knihovna objektově orientovaný návrh, avšak nepoužívá pro to žádné dodatečné nástroje. Prvním parametrem funkcí je

[11] http://www.zlib.net/

vždy struktura (objekt), mezi dvě nepominutelné patří:

- `struct archive` – abstraktní struktura archivu
- `struct archive_entry` – reprezentace hlavičky souboru či adresáře

Knihovna nabízí tři způsoby přístupu k V/V operacím. Soubor archivu je čten z paměti, z disku nebo pomocí programátorem definovaných funkcí zpětného volání. Je tedy možné dodat vlastní funkce pro otevření, čtení či zápis archivu, jeho zavření a volitelně metody pro skákání v datovém toku. Pro posun v datovém toku knihovna LibArchive rozlišuje dva způsoby: pouze dopředné skoky nebo skoky na libovolnou pozici. Dopředné skoky se využívají pro přeskočení datových oblastí, druhé typy skoků jsou ve složitějších formátech, např. u ZIP pro skok do centrální hlavičky [27].

2.4 Podobné systémy pracující s archivy

Existuje spousta dalších možností práce s archivy. V této části se zaměříme především na podobná a volně dostupná řešení. Přímo na knihovně LibArchive staví několik takových systémů, některé z nich budou dále více rozebrány [28].

2.4.1 Manažer archivů File Roller

File Roller[12] je v současné době hlavním správcem archivů grafického prostředí plochy GNOME. Funguje jako samostatná aplikace a nelze tak procházet archivy přímo z nějakého souborového manažeru. Nicméně např. Nautilus umožňuje s její pomocí alespoň vytváření a rozbalování archivů přes kontextové nabídky. Aplikace podporuje širokou škálu formátů a umožňuje čtení, úpravy i vyváření archivů.

File Roller přímo neobsahuje kompresní algoritmy, jedná se pouze o grafické rozhraní pro jednoduché konzolové nástroje jako např. `tar` či `zip`. Spektrum úprav archivů se odvíjí od možností těchto utilit. S touto koncepcí se pojí množství problémů a naneštěstí se zdá, že se vývoj v roce 2009 zastavil [29]. Mimo problémů způsobených rozdílností API jednotlivých nástrojů se potíže vyskytují hlavně při úpravách archivů [30]. Některé nástroje si vytváří dočasné soubory na různých místech, jiné zase upravují soubory přímo. Při neočekávaných situacích (např. nedostatek místa na disku) se nezřídka stává, že se nenávratně ztratí celý archiv nebo zůstane schovaný v nějakém dočasném adresáři.

2.4.2 Správce archivů Ark

Hlavním správcem grafického prostředí plochy KDE je *Ark*[13]. Pomocí rozšíření lze tuto aplikaci integrovat do *Konquororu*[14], hlavního správce souborů tohoto prostředí. Podporuje spoustu formátů archivů včetně podpory zápisu.

Na rozdíl od File Rolleru využívá aplikace ke své činnosti mimo konzolových nástrojů knihovnu *LibArchive* a funkce aplikace *KArchiver*. Nad knihovnou LibArchive jsou zde v C++ implementované operace pro úpravy archivů. Tato funkčnost je dosažena pomocí vytváření dočasného souboru.

[12] http://fileroller.sourceforge.net/
[13] http://userbase.kde.org/Ark
[14] http://www.konqueror.org/

2.4.3 FUSE Archive Mount

FUSE Archive Mount je technologicky velmi podobné řešení systému GVFS s démonem pro práci s archivy [31]. Je postaven nad virtuálním souborovým systémem *FUSE*. To je modul jádra operačních systémů založených na Unixu, který umožňuje uživatelům bez administrátorských práv vytváření vlastních souborových systémů. Pro FUSE existuje velmi mnoho podobných projektů. Ty obvykle podporují jen jeden formát (mj. Rar2FS, Fuse-zip, Tar pipe FS), v opačném případě však nepodporují zápis (např. UnpackFS, AVFS).

Nástroj Archive Mount využívá knihovnu *LibArchive*, díky které umí pracovat s různými formáty. Má podporu i pro úpravy archivů. Změny se zaznamenávají do virtuálního souborového systému a promítnou se do archivu až při jeho odpojení. Díky tomu jsou úpravy velmi rychlé, ale toto řešení může snadno vést ke ztrátám dat. Chyba se vyskytne například při nakopírování souborů do archivu a jejich následném odstranění z disku ještě před odpojením samotného archivu.

2.4.4 KIO a jeho rozšíření

KDE Input/Output, krátce *KIO*, je abstrakce souborového systému podobně jako GIO [32, 33]. Tento systém je součástí programátorských knihoven *KDELibs*[15] prostředí KDE a obsahuje moduly pro práci s různými protokoly včetně několika druhů archivů (AR, TAR a ZIP). KIO tak poskytuje jednotné API v jazyce C++ pro práci se souborovými systémy. Tento systém mimo jiné využívá manažer souborů *Konqueror*.

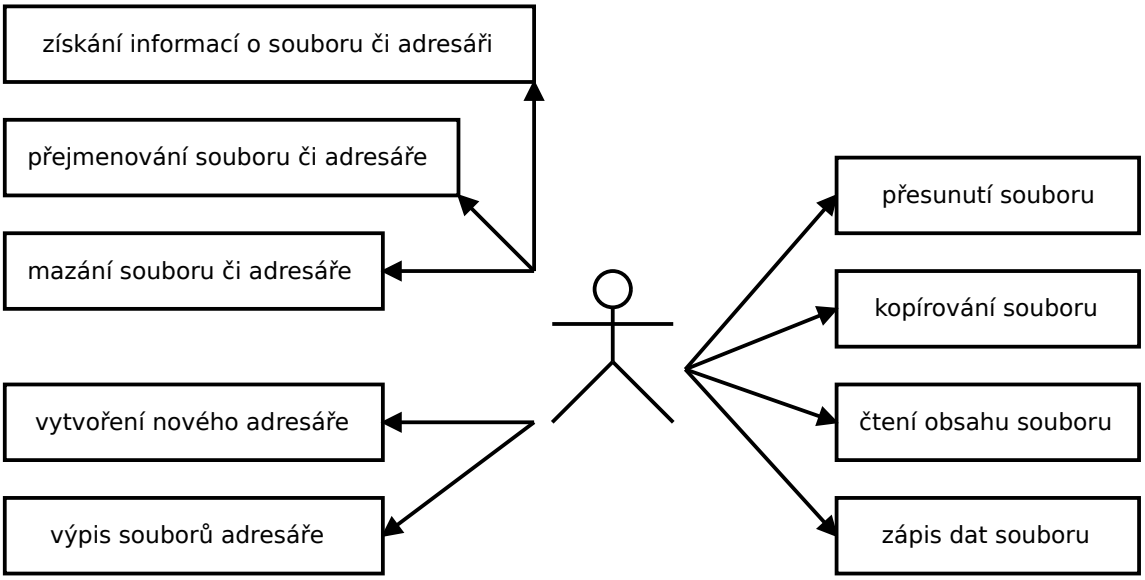
Podporované formáty archivů jsou implementovány s pomocí knihovny *ZLib*. Funkčnost zápisu je pouze pro formáty ZIP a TAR. Jejich implementace poskytují výbornou výkonnost, bohužel za cenu možné ztráty dat. Úpravy u formátu TAR jsou prováděny na rozbaleném archivu na disku. Naopak u ZIP jsou operace bez zálohy prováděny přímo na souboru archivu. Dalším nedostatkem je, že při mazání souborů z archivu ZIP dochází pouze k odmazání informací z hlaviček a data tam fyzicky zůstávají.

[15] http://api.kde.org/

Kapitola 3

Návrh na rozšíření démona

Rozšíření podpory archivů v systému GVFS vychází z požadavků na stránkách *GNOME Bugzilla* [3]. Práci je možné implementovat v rámci již existujícího řešení nebo jako zcela nový projekt. Byla zvolena první možnost, protože stávající kód je vhodně navržen a odladěn. Oproti současnému stavu bude rozšíření podporovat operace pro vytváření nových adresářů, přesouvání, kopírování, mazání souborů či adresářů a úpravy obsahu souborů (viz obrázek 3.1). Tato práce se zaměřuje pouze na funkčnost samotného démona, protože knihovna LibArchive již podporuje všechny běžně používané formáty a aktivně se vyvíjí.



Obrázek 3.1: Návrh rozšíření podporovaných operací.

Hlavní požadavkem na navrhované rozšíření je zejména zabezpečení proti ztrátě dat. Mělo by se vyrovnat s nedostatkem místa na disku nebo nesprávným odpojením démona. Musí se například počítat s tím, že uživatelé nemají ve zvyku softwarově odpojovat svá přenosná paměťová zařízení. V této kapitole budou na začátku diskutovány možné způsoby optimalizace v komunikaci s knihovnou LibArchive a způsoby úpravy archivů. Dále budou navrženy potřebné metody, které je nutné implementovat.

3.1 Minimalizace komunikace s LibArchive

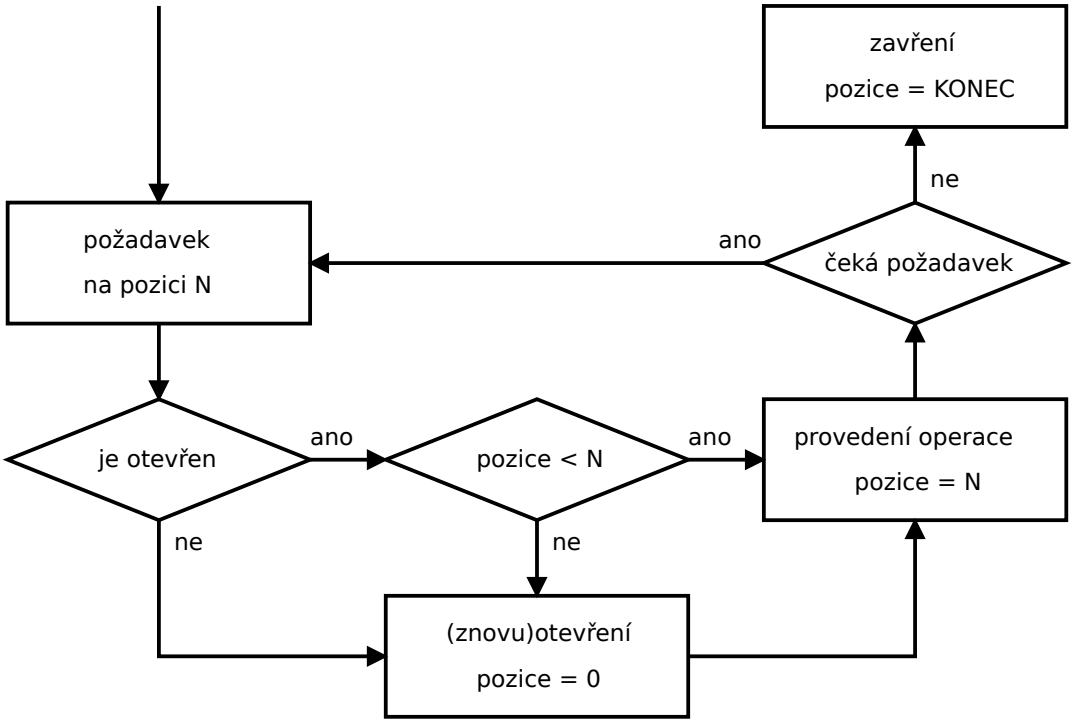
Je zapotřebí minimalizovat komunikaci knihovny LibArchive a systému GVFS. LibArchive poskytuje pouze sekvenční přístup na rozdíl od systému GVFS, který je postaven na náhodném přístupu. Démon pro práci s archivy již využívá různé optimalizace: virtuální strom souborů a paralelní provádění operací, které se běžně používá v celém systému GIO. Průzkum existujících řešení v sekci 2.4 nabízí několik přístupů:

- práce na rozbaleném archivu (zápis při odpojení)
- provádění úprav pouze ve virtuálním stromu (zápis při odpojení)
- úpravy přímo v souboru archivu

První dva způsoby poskytují vysokou efektivitu, bohužel obě dvě mají problém s bezpečností dat. Jestliže nebude archiv správně odpojen, nebudou uloženy provedené změny. První z nich má dále nevýhodu ve velké paměťové náročnosti, druhé v možném odstranění ještě nezapsaných souborů. Třetí řešení není moc rychlé, na druhou stranu při neodpojení démona budou data zachována v pořádku, proto bude tento způsob využit. Dále budou rozebrány další možné optimalizace vycházející z tohoto řešení.

3.1.1 Neuzavírání archivu po každé operaci

Jednou z možností je neuzavírat archiv dokud přichází další požadavky pro následující soubory z archivu, jak je znázorněno na obrázku 3.2. Pokud další požadavek nepřijde nebo není na některý z následujících souborů, musí být archiv zavřen včetně zapsání případných změn.



Obrázek 3.2: Zjednodušený algoritmus optimalizace.

Situace však není až tak jednoduchá. Démon totiž neví, zda přijde další požadavek či nikoli, dokud stávající operace neskončí. Tento problém se dá řešit například časovačem. To ovšem není nejšťastnější řešení, protože po oznámení konce asynchronní operace ještě nebudou data uložena a je tu opět riziko ztráty dat. Další nevýhoda tohoto způsobu je, že vyžaduje součinnost aplikace, aby při operacích nad více soubory posílala požadavky ve správném pořadí. Nicméně pořadí v archivu může zjistit z propagovaného čísla i-uzlu. Dále aplikace nesmí po každé operaci kontrolovat její výsledek (výpisem adresáře nebo informací o souboru) jako to dělá např. správce souborů Nautilus. Pokud klientská aplikace nebude přizpůsobena pro práci s tímto démonem, optimalizace mohou mít opačný efekt. Hlavním důvodem, proč nebylo toto řešení vybráno, je skutečnost, že vylučuje paralelní přístup využívaný v systému GIO.

3.1.2 Vícesouborové operace v GIO

Další z možných optimalizací je zavedení vícesouborových operací v GIO. Poté by bylo možné při jednom průchodu archivem provést několik změn, což by poskytnulo rychlou práci a zároveň požadovanou bezpečnost dat. Avšak funkce pro práci s více soubory naráz již byly v systému GnomeVFS, které bylo právě pro své komplikované rozhraní nakonec nahrazeno systémem GIO [9, 34]. Tato změna by navíc obnášela složité úpravy interního programátorského rozhraní GIO. Navrhované řešení bylo tedy ze zmíněných důvodů také zavrhnuto.

3.2 Úprava archivu pomocí LibArchive

Podpora pro úpravu archivů v knihovně LibArchive chybí. Nezávisle na použité optimalizaci musí být tato vlastnost implementována. Jednou z možností je úprava aplikačního rozhraní knihovny. Tyto změny by však porušovaly filosofii knihovny a obnášely velké strukturální změny způsobené rozdílnou vnitřní strukturou u různých formátů archivů. Jednodušší možnost se nabízí při využití existujícího API, podobně jako v případě existujících řešení zmíněných v sekci 2.4.

V současné době asi neexistuje lepší způsob, než postupně číst původní archiv a zapisovat ho s příslušnými úpravami do dočasného souboru [35]. Zda se změna provede u konkrétního souboru, je většinou určeno podle cesty. V případě adresářů se musí dát pozor, že jeho cesta je součástí jmen souborů, které obsahuje. Operace nad složkami se tak musí provést na více místech. Po dokončení všech úprav se nahradí originální soubor, což zároveň poskytuje ochranu proti ztrátě dat. Dočasný soubor je kvůli efektivitě vhodné umístit na stejný diskový oddíl či paměťové medium.

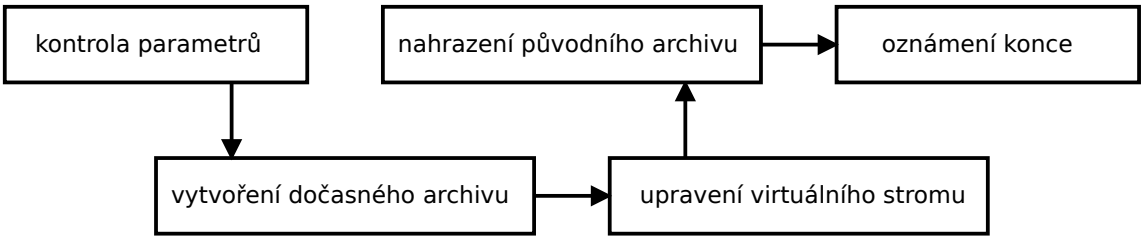
3.3 Metody pro rozšíření funkčnosti démona

Rodičovská třída `GVfsBackend` obsahuje deklarace metod pro práci s daty, které můžeme využít pro rozšíření funkčnosti. Pro splnění cílů nás budou zajímat zejména funkce:

- `make_directory` – vytvoření nového adresáře
- `set_display_name` – přejmenování souboru či adresáře
- `delete` – smazání souboru či adresáře

- `move` – přesunutí souboru v rámci archivu
- `create` – vytvoření nového souboru a otevření pro zápis
- `append` – otevření souboru pro zápis na konec
- `replace` – otevření souboru pro zápis nového obsahu
- `write` a `close_write` – zapsání dat do otevřeného souboru a uzavření

Většina z těchto funkcí má podobnou výstavbu, která je znázorněna na obrázku 3.3. Nejprve je potřeba zkontrolovat vstupní parametry, provést požadovanou úpravu v archivu, tuto změnu následně zanést i do virtuálního stromu, nahradit původní archiv a současně informovat hlavního démona GVFS o konci operace. Jednotlivé fáze musí být v tomto pořadí, aby se démon nedostal do chybového stavu. V případě, že v některém z úseků nastane chyba, oznámí se hlavnímu démonu a dále se nepokračuje. Konec operace se ohlašuje funkcemi `g_vfs_job_succeeded` a `g_vfs_job_failed_from_error`.



Obrázek 3.3: Zjednodušené schéma struktury operací.

Návrh jednotlivých metod a jejich požadované chování vychází z funkcí GIO zmíněných v části 2.1.3. Pro oznamování chyb jsou využita chybová hlášení popsaná v sekci 2.1.4. Při přesunu či kopírování je ošetření chyb poměrně komplikované, proto je shrnuté v tabulce 3.1 [12]. Operace mohou být spouštěny ve více vláknech, proto musí být všechny funkce opatřeny zámky, aby se démon nedostal do chybového stavu při práci s virtuálním stromem nebo při zápisu do souboru. Pro samotnou implementaci těchto operací bude vhodné vytvořit pomocné funkce popsané dále.

zdroj	*cíl*	*bez přepisování*	*s přepisováním*
neexistuje	neexistuje	soubor neexistuje	soubor neexistuje
neexistuje	soubor	soubor neexistuje	soubor neexistuje
neexistuje	adresář	soubor neexistuje	soubor neexistuje
soubor	neexistuje	bez chyby	bez chyby
soubor	soubor	soubor existuje	bez chyby
soubor	adresář	soubor existuje	jedná se o adresář
adresář	neexistuje	rekurzivní operace	rekurzivní operace
adresář	soubor	soubor existuje	rekurzivní operace
adresář	adresář	soubor existuje	operace slučování

Tabulka 3.1: Chybová hlášení ve funkcích pro přesun či kopírování.

V již existující funkci `try_query_fs_info` pro informace o virtuálním souborovém systému musí dojít ke změně, aby archiv GIO věděl, že systém není pouze pro čtení. Toho lze

docílit pomocí operace `g_file_info_set_attribute_boolean` s hodnotou `FALSE` pro atribut `G_FILE_ATTRIBUTE_FILESYSTEM_READONLY`.

3.4 Funkce pro manipulaci s archivem

Pro manipulaci s archivem bude zapotřebí rozšířit strukturu `GVfsArchive` popsanou v sekci 2.2.2. Chybí zde objekt `GFile` pro reprezentaci dočasného souboru archivu, struktura archivu pro zápis a výstupní datový tok. Tyto tři položky umožní práci s pomocným archivem.

Většina nových operací démona bude kopírovat části archivu. Proto bude vhodné dále vytvořit funkce pro kopii dat a kopii části archivu. Částí se rozumí kopie tolika souborů, dokud nebude objeven soubor s požadovanou cestou, nad kterým bude potřeba provést nějaká činnost. Kvůli některým operacím bude nutné dále přidat funkci pro kopii informací o souboru z virtuálního stromu do hlavičky v archivu.

V současné době se chyby při práci s knihovnou LibArchive ošetřují jinak, než chyby funkcí GIO. Pro jednotnou práci by všechny tyto funkce v případě chyby v LibArchive měly nastavit objekt `GError` ve struktuře archivu. Pro ulehčení nebudou funkce při nastavené chybě nic vykonávat a při volání vrátí pouze tu samou chybu. Po této úpravě se tak nebudou muset kontrolovat chyby po každém příkazu a díky tomu se zpřehlední a zjednoduší kód. Aby však byla dosažena jednotná správa chyb, je nutné zapouzdřit i funkce LibArchive pro čtení hlavičky či dat a obdobně pro jejich zápis.

3.5 Funkce pro práci s virtuálním stromem

S novou funkčností démona rostou požadavky i na virtuální strom souborů. Aby bylo možné jednoduše manipulovat se stromem, je potřeba přidat do struktury `ArchiveFile` ukazatel na rodičovský uzel. Tuto změnu je samozřejmě potřeba zanést do již existujících funkcí. Při nastavování informací o souboru je také nutné povolit provádění nových operací. Na obrázku 3.4 se nalézají atributy, které je potřeba změnit.

```
g_file_info_set_attribute_boolean (info,
                                   G_FILE_ATTRIBUTE_ACCESS_CAN_WRITE,
                                   TRUE);
g_file_info_set_attribute_boolean (info,
                                   G_FILE_ATTRIBUTE_ACCESS_CAN_DELETE,
                                   TRUE;
g_file_info_set_attribute_boolean (info,
                                   G_FILE_ATTRIBUTE_ACCESS_CAN_RENAME,
                                   TRUE);
```

Obrázek 3.4: Povolení nových operací u souborů.

Nově je potřeba implementovat metody pro přejmenování uzlů, přesun uzlů, odstranění jednoho konkrétního uzlu a změnu informací o uzlu. Při přejmenovávání se hodnota musí změnit nejen v názvu, ale i v informacích.

3.6 Podpora pro vytváření nových archivů

Přímo v aplikačním rozhraní GIO nejsou vhodné funkce pro vytváření nových archivů. Pro rychlou integraci např. se správcem Nautilus by bylo možné využít mechanismus šablon souborů. Mezi dostupné šablony by stačilo přidat několik prázdných archivů různých formátů a uživatel by si pak několika kliknutími mohl vytvořit nový archiv. Toto řešení však nepodporuje nějaká složitější nastavení. Kromě tohoto způsobu existují další dvě možnosti:

- zavedení speciálního tvaru URI
- využití parametrů příkazové řádky

Vytvoření speciálního identifikátoru by bylo sice možné, ale hlavní nevýhodou však je, že by pro uživatele nebylo příliš srozumitelné a bylo by pouze dočasné. Druhé nabízené řešení by navenek zůstalo skryté a URI by bylo neměnné. Pomocí parametrů lze přidat libovolné nastavení, pro které by do budoucna bylo vhodné implementovat dialogové okno s dostupnými možnostmi.

Kapitola 4

Implementace navrženého rozšíření

Tvorba rozšíření vychází z návrhu v kapitole 3. Práce však nebyla úplně přímočará, proto jsou dále popsány některé implementační detaily, které bylo potřeba při vývoji rozšíření vyřešit. Na začátek je zde rozebráno několik problémů v komunikaci s knihovnou LibArchive a také důvod, proč démon pro práci s archivy neumožňuje úpravy existujících souborů. Dále jsou zmíněny objevené a reportované chyby v LibArchive i v démonu včetně jejich řešení. Na závěr je zvolen nástroj pro generování dokumentace.

4.1 Zarovnávání bloků v LibArchive

Bez nějakého dodatečného nastavování knihovny LibArchive docházelo k častým selháním při čtení archivu. Po několikáté úpravě archivu démon končil s chybami např. `damaged tar archive` nebo `gzip compression failed`. Dalším příznakem bylo, že malý archiv po úpravě v systému GVFS několikanásobně zvětšil svoji velikost.

Chyba byla odhalena díky zkoumání zdrojových kódů pro zápis na disk v systému LibArchive. Knihovna má totiž v základním stavu nastavené zarovnávání dat na bloky o velikosti 10240 bajtů, což je samozřejmě dobré pro efektivní čtení [36]. Toto zarovnávání je i pro poslední blok na stejnou velikost. Nicméně zapisovaná data mohla být na konci zarovnána ještě systémem GIO [37], což způsobovalo výše zmíněné problémy. Řešením je nezarovnávat data v posledním bloku knihovnou LibArchive. To je možné nastavit pomocí funkce `archive_write_set_bytes_in_last_block`.

4.2 Problémy při čtení ZIP archivů

Tento problém postihuje pouze archivy formátu ZIP. Náznaky byly objeveny již při studiu zdrojových kódů LibArchive, kdy přiložený nástroj `bsdcpio` nedovedl rozbalit soubory jednoho archivu. Zádrhel byl v hlášené nulové velikosti dat obsažených souborů knihovnou LibArchive, a proto neprobíhalo jejich kopírování. Chyba byla nahlášena a vývojáři následně opravena [38, 39]. Oprava spočívala pouze v tom, že i přes nezadanou velikost souborů v archivu došlo k jejich vybalení na disk.

Úplný původ tohoto problému se skrývá ve specifikaci ZIP archivů. Při procházení archivu jsou čteny postupně lokální hlavičky souborů, které nemusí obsahovat velikost souboru. Ta se může nacházet až v centrálním adresář na konci archivu [27]. LibArchive tak při sekvenčním přístupu nezná velikost.

Následně však byl objeven související problém v démonu pro práci s archivy, který byl také nahlášen [40]. Zde však tento problém působí mnohem větší problémy. Jeden z nich je, že chybějící velikost se předává dále aplikacím, které se s tím obvykle nedokáží vyrovnat (např. správce souborů Nautilus). Dále pak tento nedostatek neumožňuje úpravu archivů, protože před zapsáním dat do dočasného archivu je potřeba znát jejich velikost. Problém byl nejprve vyřešen při tvorbě virtuálního stromu přečtením všech dat souboru pro určení velikosti. Tato oprava je ovšem velmi výpočetně náročná. V implementovaném rozšíření je nakonec využita funkce zpětného volání pro skoky v archivu, která dovoluje nahlédnout do centrálního registru ZIP archivu. Nicméně toto řešení předpokládá možnost posunu v datovém toku.

4.3 Kopie nastavení v LibArchive

Pro úpravy archivů je potřeba přenést nastavení ze čteného archivu do zapisovaného. LibArchive poskytuje různé funkce pro manipulaci s použitými formáty, filtry a dalšími parametry archivu. Je možné jednoduše zjistit kód nastaveného formátu a s jeho pomocí snadno nastavit stejný formát jinému archivu. Existuje zde také podobná metoda pro zjištění kódů použitých filtrů, pro jejich nastavení však bohužel ne. Proto byla chybějící funkce `archive_write_add_filter` implementována a zařazena do vývojové verze knihovny [35]. Její kopie s podmíněnými makry byla vložena i do kódů samotného démona, aby byla zachována podpora pro co nejstarší verzi LibArchive.

Větší problém nastal s kopií vlastností jednotlivých filtrů a formátů. Knihovna poskytuje funkce pro jejich nastavení, bohužel podpora pro jejich zpětné zjištění neexistuje. Není tak možné zjistit např. velikost použité komprese. Díky dobrému návrhu knihovny by přidání této chybějící funkcionality nebylo složité, bylo by však značně časově náročné. Muselo by se rozšířit aplikační rozhraní knihovny i vnitřní struktury a následně chybějící funkci doimplementovat do všech formátů a filtrů. Jelikož knihovna zatím umožňuje nastavit pouze minimum vlastností, byla zvolena jednodušší cesta. Pro všechny součásti je nastavena maximální možná komprese pomocí parametru `compression-level=9` funkce `archive_write_set_options`.

4.4 Neznámá velikost zapisovaného souboru

Metody třídy démona pro zápis a úpravu souborů fungují na principu datových toků zmíněných v části 2.1.2. Funkce zapisují příchozí data, u kterých není předem známa jejich velikost a ani neexistuje způsob, jak ji zjistit. Naopak operace v knihovně LibArchive potřebují předem vědět přesnou velikost a bez ní nebudou pracovat. Problém se konkrétně týká funkcí `append`, `create` a `replace`, které kromě úpravy obsahu souborů slouží k vytváření nových souborů či kopírování.

Nabízí se možné řešení s pomocí dalšího dočasného souboru. Pro určení velikosti mohou být data nejdříve nakopírována do pomocného souboru a až následně zapsána do dočasného souboru archivu. Při přesunu velkého souboru je toto řešení paměťově i výpočetně náročné a proto nebylo implementováno.

Aby bylo alespoň možné pomocí démona soubory do archivu přidávat, místo těchto funkcí byla použita metoda `push`. Ta původně byla do GVFS přidána pouze za účelem tvorby knihoven a neměla sloužit pro vlastní práci démonů, protože nepracuje na principu datových toků. Funkce obdrží samotnou cestu k souboru a stará se kompletně o jeho přenos,

díky tomu není problém zjistit velikost souboru a následně zkopírovat do dočasného archivu bez dalších mezikroků. Tato funkce je bohužel omezena pouze na přenos souborů z lokálního disku.

4.5 Propagování jména souboru

Při implementaci metody pro přejmenování souboru `set_display_name` nestačí přejmenovat soubor jen v archivu a virtuálním stromu démona. Změnu jména je potřeba oznámit systému GVFS, jinak se systém dostane do chybového stavu. Nezaregistruje provedenou úpravu, bude stále zobrazovat původní jména a veškeré operace nad těmito soubory budou končit chybou. Po úspěšném přejmenování je nutné použít funkce pro propagaci změněných informací o souboru `gvfs_file_info_populate_default` a oznámení nového jména `g_vfs_job_set_display_name_set_new_path`.

4.6 Nástroj na generování dokumentace

Zdrojové kódy práce bylo zapotřebí zdokumentovat. Pro generování dokumentace API se v projektech Gnome používá nástroj *GTK-Doc*[1], který je určen pro jazyk C. Nebyl však navržen jako univerzální nástroj pro tvorbu dokumentace, a proto jeho nastavení i integrace do projektu je komplikovaná. Na druhou stranu umí na rozdíl od jiných produktů zacházet např. se systémem GObject. V GVFS není zakomponován, protože se jedná o interní nedokumentované programové rozhraní.

Speciální vlastnosti systému GTK-Doc nejsou pro démona potřebné a proto bylo zvoleno jednodušší řešení, kterým je univerzální nástroj *Doxygen*[2]. Ten poskytuje snadný způsob pro vytvoření dokumentace různých formátů (mj. HTML, Latex, PDF, RTF, PostScript) z mnoha programovacích jazyků (např. C, C++, C#, Java, Python, VHDL).

[1] http://www.gtk.org/gtk-doc/
[2] http://www.stack.nl/~dimitri/doxygen/

Kapitola 5

Automatická testovací sada

Démon byl při rozšiřování průběžně testován pomocí nástrojů příkazové řádky, které poskytuje GVFS (viz sekce 2.2.1). Tato kapitola však popisuje návrh a tvorbu testovací sady pomocí funkcí GIO, aby tak testy mohly být snadno distribuovány se zdrojovými kódy démona a používány pro regresní testování v případě dalšího vývoje. Nejprve bude rozebráno rozhraní knihovny GLib pro testování a některé implementační problémy, které bylo zapotřebí při tvorbě testů vyřešit. Následně bude popsána testovací sada.

5.1 Testovací rozhraní GLib

Programátorská knihovna GLib poskytuje testovací rozhraní pro psaní a údržbu testů v závislosti na testovaném zdrojovém kódu [41]. Její API je navrženo na ustálených principech nalézajících se např. v knihovnách JUnit, NUnit či RUnit, které vychází z testovacích knihoven jazyku Smalltalk.

Rozhraní umožňuje vytvářet z jednotlivých testů celé testovací případy a ty řadit do skupin. Stará se o jejich spouštění a detailní ladící výstupy. Na obrázku 5.1 vidíme inicializaci testovacího rozhraní pomocí parametrů příkazové řádky, zařazení testovacího případu ve funkci `test` do skupiny testů `archiv` a následné spuštění.

```
g_test_init (&argc, &argv, NULL);
g_test_add_func ("/archiv/test", test);
g_test_run ();
```

Obrázek 5.1: Příklad testovacího rozhraní GLib.

Samotné testovací funkce obsahují množství kontrol, které v případě neúspěchu ukončí provádění celé skupiny testů. Tyto kontroly jsou realizovány s využitím různých variant klasického příkazu `assert` pro porovnávání:

- číselných hodnot s plovoucí řádovou čárkou – `g_assert_cmpfloat`
- celočíselných hodnot – `g_assert_cmpint`, `g_assert_cmpuint` a `g_assert_cmphex`
- řetězců – `g_assert_cmpstr`
- chybových hlášení – `g_assert_error` a `g_assert_no_error`

Tyto varianty přináší výhody v podrobných chybových výpisech, které obsahují porovnávané hodnoty a napomáhají tak opravování chyb.

5.1.1 Převod asynchronních funkcí na synchronní

Při implementaci testů bylo zapotřebí převést asynchronní funkce GIO na synchronní. Na obrázku 5.2 je převedena funkce pro připojení virtuálního svazku, avšak princip je prakticky stejný i pro ostatní funkce. Ve funkci pro připojení `mount` se zavolá asynchronní funkce `g_file_mount_enclosing_volume` a předá se ji ukazatel na funkci zpětného volání `callback` spolu se strukturou `struct data`. Funkce `callback` je vyvolána po dokončení asynchronní operace, pomocí předané struktury vrátí případné chyby a oznámí dokončení pomocí proměnné `finished`. Na tuto změnu se čeká ve smyčce pomocí operace `g_main_context_iteration`, která umožní zpracovávání důležitějších událostí a nezatěžuje tak procesor [42].

```
struct data
{
  gboolean finished;
  GError *error;
};

void callback (GObject *o, GAsyncResult *r, gpointer callback_data)
{
  struct data *data = callback_data;

  g_file_mount_enclosing_volume_finish (G_FILE (o), r, &data->error);
  data->finished = TRUE;
}

GError *mount (GFile *f)
{
  struct data data = {FALSE, NULL};

  g_file_mount_enclosing_volume (f, 0, NULL, NULL, callback, &data);
  while (!data.finished)
    g_main_context_iteration (NULL, TRUE);

  return data.error;
}
```

Obrázek 5.2: Převod asynchronní funkce na synchronní.

I přes tento převod nebylo možné ihned pracovat s daným objektem, v tomto případě s připojeným archivem, a proto byl nakonec zahrnut ještě časovač. Ten byl přidán pomocí funkce `g_timeout_add` na dobu 2 vteřin. Doba čekání byla experimentálně zjištěna a z bezpečnostních důvodů zdvojnásobena. Jedná se pravděpodobně o interní chybu GVFS, která již byla ve vývojové větvi opravena.

5.2 Návrh a implementace testů

Testy vycházejí z návrhu démona popsaného v kapitole 3. Způsob testování je inspirován již existujícími testy obsaženými v systému GIO. Využívá veřejné aplikační rozhraní GIO, které se na rozdíl od vnitřního GVFS API příliš nemění. Princip testování je následující. Nejprve je připojen démon na dočasný archiv, dále se provede požadovaná operace a zkontroluje se její výsledek. Poté je démon odpojen a znovu připojen a dochází k dodatečné kontrole, zda při opětovném připojení zůstaly změny zachovány. Tímto způsobem se kontrolují dvě věci: zda lze archiv po provedených úpravách znovu přečíst a zda byly provedené úpravy zapsány nejen do virtuálního stromu, ale i do souboru archivu.

Tyto testy nemají za cíl kontrolovat správnost formátu a jiné náležitosti archivů. Za tímto účelem knihovna LibArchive obsahuje vlastní testovací sady, které pokrývají většinu její funkcionality.

Jednotlivé skupiny testů by měly být ze zadání jednotkové, tedy v ideálním případě nezávislé. To však díky návrhu démona není úplně možné, proto jsou skupiny testů vykonávány v daném pořadí. Pokud funguje připojování a odpojování démona, má smysl testovat další skupiny atd. Testovací sada obsahuje v tomto sledu volané skupiny testů:

- připojení a odpojení démona
- získání informací o souboru či adresáři
- výpis souborů adresáře
- čtení obsahu souboru
- přejmenování souboru či adresáře
- vytvoření nového adresáře
- mazání souboru či adresáře
- přesunutí souboru v rámci archivu
- kopírování souboru z lokálního disku do archivu

Implementované testy však nemohou být automaticky spouštěny při každém překladu GVFS. Systém pro svůj běh vyžaduje běžící systémové služby (např. D-Bus), které na překladových serverech nejsou spuštěny. Nicméně tyto testy jsou velmi vhodné pro regresní testování při dalším vývoji.

Kapitola 6

Závěr

Úkolem této bakalářské práce bylo rozšířit podporu archivů v systému GVFS. Nejprve byl zmapován současný stav systému a jeho možnosti, dále byly prozkoumány potřebné technologie a podobná řešení. Na základě získaných informací bylo navrženo rozšíření démona pro práci s archivy, které v sobě zahrnuje možnost vytvářet nové archivy a dále přesouvat, kopírovat, přejmenovávat, upravovat a mazat obsažené soubory a adresáře. Nakonec bylo dané rozšíření implementováno a důkladně otestováno navrženými testy. Podpora pro úpravu souborů implementována nebyla z důvodu vysoké paměťové náročnosti řešení, která je způsobena rozdílností aplikačních rozhraní knihovny LibArchive a démona GVFS (viz sekce 4.4).

Při vývoji rozšíření byla obohacena i knihovna LibArchive o novou funkčnost, která je již obsažena ve vývojové verzi. Mimo to byly objeveny a nahlášeny nové chyby v kódech knihovny a následně ve spolupráci s vývojáři odstraněny [38, 39]. Na základě těchto chybových hlášení byl nalezen podobný problém i v démonu pro práci s archivy, který byl touto prací částečně opraven [40].

Vytvořený nástroj v kombinaci s podporovanými správci souborů přináší jednoduchou cestu pro práci s archivy. Ve spolupráci s firmou Red Hat Czech, s. r. o. by v blízké době mělo být rozšíření zahrnuto do oficiálních zdrojových kódů GVFS. V grafickém prostředí plochy GNOME má pak tato práce šanci v součinnosti se správcem souborů Nautilus nahradit program File Roller. Z hlediska dalšího vývoje tohoto nástroje bude zapotřebí zajistit podporu pro úpravu souborů. Nezbytné pro správce souborů Nautilus však bude vytvořit dialogové okno pro tvorbu nových archivů, což je již vypsané jako téma diplomové práce.

Přáním společnosti Red Hat Czech, s. r. o. bylo upravit stávající kód, avšak pro budoucí vývoj by bývalo bylo lepší vytvořit nové třídy pro práci s virtuálním stromem a s archivem. Třída pro práci s archivy by se poté dala používat samostatně i v jiných projektech. Další možností by bylo tuto třídu implementovat přímo jako rozšíření pro knihovnu LibArchive.

Literatura

[1] Mount zip files from the command line using gvfs-mount. *Ubuntu Forums* [online]. Canonical Ltd., 2010-04-13 [cit. 2012-04-15]. Dostupné z: http://ubuntuforums.org/showthread.php?t=1207096

[2] Zip file gvfsd-archive/archive mounter read write mode. *Ubuntu Forums* [online]. Canonical Ltd., 2011-06-26 [cit. 2012-04-15]. Dostupné z: http://ubuntuforums.org/showthread.php?t=1639312

[3] Bug 589617: gvfsd-archive write support. *GNOME Bugzilla* [online]. The GNOME Project, 2011-12-17 [cit. 2012-04-15]. Dostupné z: https://bugzilla.gnome.org/show_bug.cgi?id=589617

[4] GLib Overview. *GLib Reference Manual: for glib 2.32.1* [online]. The GNOME Project [cit. 2012-04-16]. Dostupné z: http://developer.gnome.org/glib/2.32/glib.html

[5] GLib. *Wikipedia: the free encyclopedia* [online]. Wikimedia Foundation, 2011-03-01 [cit. 2012-04-16]. Dostupné z: http://en.wikipedia.org/wiki/GLib

[6] GObject: introduction. *GObject Reference Manual: for gobject 2.32.1* [online]. The GNOME Project [cit. 2012-04-16]. Dostupné z: http://developer.gnome.org/gobject/stable/pr01.html

[7] GObject. *Wikipedia: the free encyclopedia* [online]. Wikimedia Foundation, 2012-13-02 [cit. 2012-04-16]. Dostupné z: http://en.wikipedia.org/wiki/GObject

[8] Type Information: the glib runtime type identification and management system. *GObject Reference Manual: for gobject 2.32.1* [online]. The GNOME Project [cit. 2012-05-07]. Dostupné z: http://developer.gnome.org/gobject/2.32/gobject-Type-Information.html

[9] KELLNER, Christian. *GIO Migration Guide* [online]. Lanedo GmbH, 2009-12-09 [cit. 2012-04-16]. Dostupné z: http://people.gnome.org/~gicmo/gio-migration-guide/

[10] GIO (GNOME). *Wikipedia: the free encyclopedia* [online]. Wikimedia Foundation, 2011-08-31 [cit. 2012-04-16]. Dostupné z: http://en.wikipedia.org/wiki/GIO_(GNOME)

[11] GIO: introduction. *GIO Reference Manual: for gio 2.32.1* [online]. The GNOME Project [cit. 2012-04-16]. Dostupné z: http://developer.gnome.org/gio/2.32/ch01.html

[12] GFile: file and directory handling. *GIO Reference Manual: for gio 2.32.1* [online]. The GNOME Project [cit. 2012-04-23]. Dostupné z: http://developer.gnome.org/gio/2.32/GFile.html

[13] GCancellable: thread-safe operation cancellation stack. *GIO Reference Manual: for gio 2.32.1* [online]. The GNOME Project [cit. 2012-04-26]. Dostupné z: http://developer.gnome.org/gio/2.32/GCancellable.html

[14] Error Reporting: a system for reporting errors. *GLib Reference Manual: for glib 2.32.1* [online]. The GNOME Project [cit. 2012-04-23]. Dostupné z: http://developer.gnome.org/glib/2.32/glib-Error-Reporting.html

[15] GIOError: error helper functions. *GIO Reference Manual: for gio 2.32.1* [online]. The GNOME Project [cit. 2012-04-23]. Dostupné z: http://developer.gnome.org/gio/2.32/gio-GIOError.html

[16] Internationalization: gettext support macros. *GIO Reference Manual: for gio 2.32.1* [online]. The GNOME Project [cit. 2012-04-26]. Dostupné z: http://developer.gnome.org/glib/2.32/glib-I18N.html

[17] Features: gvfs. *Fedora Project: wiki* [online]. Red Hat, Inc., 2008-05-24 [cit. 2012-04-16]. Dostupné z: http://fedoraproject.org/wiki/Features/Gvfs

[18] GVFS. *Wikipedia: the free encyclopedia* [online]. Wikimedia Foundation, 2011-10-11 [cit. 2012-04-16]. Dostupné z: http://en.wikipedia.org/wiki/GVFS

[19] GARNACHO, Carlos. *Gio/GVfs* [online]. ImendioConf 2007 [cit. 2012-04-16]. Dostupné z: http://www.lanedo.com/~carlos/Talks/ImendioConf2007-Gio-GVfs.pdf

[20] LARSSON, Alexander. *GVFS: the making of a virtual filesystem* [online]. GUADEC 2007 [cit. 2012-04-16]. Dostupné z: http://people.gnome.org/~alexl/presentations/guadec2007-gvfs.pdf

[21] Chapter 12. System Utilities: introduction to gvfs. *Beyond Linux From Scratch* [online]. The BLFS Development Team, 2012-04-16 [cit. 2012-04-17] Dostupné z: http://www.linuxfromscratch.org/blfs/view/cvs/general/gvfs.html

[22] URI Functions. *GLib Reference Manual: for glib 2.32.1* [online]. The GNOME Project [cit. 2012-04-19]. Dostupné z: http://developer.gnome.org/glib/2.32/glib-URI-Functions.html

[23] KIENTZLE, Tim. *Archiving and Packaging A Survey: How I Accidentally Rewrote Tar* [online]. 2006-06-12 [cit. 2012-04-17]. Dostupné z: http://people.freebsd.org/~kientzle/libarchive/src/libarchive-slides-2006.06.12.pdf

[24] Examples. *Libarchive: C library and command-line tools for reading and writing tar, cpio, zip, ISO, and other archive formats* [online]. GitHub Inc. [cit. 2012-04-18]. Dostupné z: https://github.com/libarchive/libarchive/wiki/Examples

[25] Libarchive Formats: summary of formats supported by the library and command-line tools. *Libarchive: C library and command-line tools for reading and writing tar, cpio, zip, iso, and other archive formats* [online]. GitHub Inc. [cit. 2012-04-17]. Dostupné z: https://github.com/libarchive/libarchive/wiki/LibarchiveFormats

[26] LIBARCHIVE(3): manual page. *Libarchive: C library and command-line tools for reading and writing tar, cpio, zip, iso, and other archive formats* [online]. GitHub Inc. [cit. 2012-04-17]. Dostupné z: https://github.com/libarchive/libarchive/wiki/ManPageLibarchive3

[27] Format Zip. *Libarchive: C library and command-line tools for reading and writing tar, cpio, zip, iso, and other archive formats* [online]. GitHub Inc. [cit. 2012-04-19]. Dostupné z: https://github.com/libarchive/libarchive/wiki/FormatZip

[28] Libarchive Users: a few projects and people who use libarchive. *Libarchive: C library and command-line tools for reading and writing tar, cpio, zip, iso, and other archive formats* [online]. GitHub Inc. [cit. 2012-04-19]. Dostupné z: https://github.com/libarchive/libarchive/wiki/LibarchiveUsers

[29] News. *File Roller: an archive manager for the gnome desktop* [online]. Geeknet Inc. [cit. 2012-04-19]. Dostupné z: http://fileroller.sourceforge.net/news.html

[30] Bug List: file-roller. *GNOME Bugzilla* [online]. The GNOME Project, 2012-04-15 [cit. 2012-04-15]. Dostupné z: https://bugzilla.gnome.org/buglist.cgi?product=file-roller&bug_status=UNCONFIRMED&bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED

[31] Archive File Systems. *FUSE: Wiki* [Online]. Geeknet Inc., 2011-10-20 [cit. 2012-04-19] Dostupné z: http://sourceforge.net/apps/mediawiki/fuse/index.php?title=ArchiveFileSystems

[32] KIO: network-enabled file management. *KDE 4.0 API Reference* [Online]. KDE e.V. [cit. 2012-04-19] Dostupné z: http://api.kde.org/4.0-api/kdelibs-apidocs/kio/html/index.html

[33] KIO. *Wikipedia: the free encyclopedia* [online]. Wikimedia Foundation, 2012-02-02 [cit. 2012-04-19]. Dostupné z: http://en.wikipedia.org/wiki/KIO

[34] File Transfers: conveniently copy/move/delete files en masse. *GnomeVFS: filesystem abstraction library* [online]. The GNOME Project [cit. 2012-04-24]. Dostupné z: http://developer.gnome.org/gnome-vfs/2.24/gnome-vfs-2.0-gnome-vfs-xfer.html

[35] Read/write memory. *Google Groups: libarchive-discuss* [online]. Google, 2012-03-28 [cit. 2012-04-14]. Dostupné z: http://groups.google.com/group/libarchive-discuss/browse_thread/thread/4cc6d6cf92cbf47

[36] ARCHIVE_WRITE_BLOCKSIZE(3): manual page. *Libarchive: C library and command-line tools for reading and writing tar, cpio, zip, iso, and other archive formats* [online]. GitHub Inc. [cit. 2012-04-28]. Dostupné z: https://github.com/libarchive/libarchive/wiki/ManPageArchiveWriteBlocksize3

[37] GConverter: data conversion interface. *GIO Reference Manual: for gio 2.32.1* [online]. The GNOME Project [cit. 2012-04-28]. Dostupné z: http://developer.gnome.org/gio/2.32/GConverter.html

[38] Issue 217: make error. *Libarchive: C library and command-line tools for reading and writing tar, cpio, zip, iso, and other archive formats* [online]. Google 2011-12-30 [cit. 2012-04-28]. Dostupné z: http://code.google.com/p/libarchive/issues/detail?id=217&thanks=217&ts=1325279347

[39] Issue 218: empty contents extracting zip files with bsdcpio. *Libarchive: C library and command-line tools for reading and writing tar, cpio, zip, iso, and other archive formats* [online]. Google, 2011-12-30 [cit. 2012-04-28]. Dostupné z: http://code.google.com/p/libarchive/issues/detail?id=217&thanks=217&ts=1325279347

[40] Bug 670534: reading empty files with gvfsd-archive. *GNOME Bugzilla* [online]. The GNOME Project, 2012-02-21 [cit. 2012-04-28]. Dostupné z: https://bugzilla.gnome.org/show_bug.cgi?id=670534

[41] Testing: a test framework. *GLib Reference Manual: for glib 2.32.1* [online]. The GNOME Project [cit. 2012-04-14]. Dostupné z: http://developer.gnome.org/glib/2.32/glib-Testing.html

[42] The Main Event Loop: manages all available sources of events. *GLib Reference Manual: for glib 2.32.1* [online]. The GNOME Project [cit. 2012-04-23]. Dostupné z: http://developer.gnome.org/glib/2.32/glib-The-Main-Event-Loop.html

Kapitola 7

Obsah přiloženého CD

Na přiloženém CD se nachází níže uvedené soubory a složky:

- `/doxygen/` – dokumentace rozhraní démona (vygenerováno nástrojem Doxygen)
- `/gvfs/` – upravené zdrojové kódy systému GVFS včetně testů (GIT repozitář)
- `/latex/` – kompletní zdrojové kódy této technické zprávy
- `/libarchive/` – upravené zdrojové kódy knihovny LibArchive (GIT repozitář)
- `/virtualbox/` – Fedora 17 Beta s upraveným démonem (disk pro VirtualBox)
- `/gvfs.diff` – záplata pro rozšíření podpory archivů v systému GVFS
- `/libarchive.diff` – záplata pro přidání nové funkčnosti do knihovny LibArchive
- `/README.txt` – návod na vyzkoušení démona
- `/xholyo00.pdf` – elektronická verze této technické zprávy